Fuzzy Adaptive Particle Swarm Optimization

Yuhui Shi
EDS Embedded Systems Team
1401 E. Hoffer Street
Kokomo, IN 46902, USA
Yuhui.shi@eds.com

and

Russell C. Eberhart
Department of Electrical and Computer Engineering
Purdue School of Engineering and Technology
799 W. Michigan Street
Indianapolis, IN 46202
Eberhart@engr.iupui.edu

Abstract

In this paper, a fuzzy system is implemented to dynamically adapt the inertia weight of the particle swarm optimization algorithm (PSO). Three benchmark functions with asymmetric initial range settings are selected as the test functions. The same fuzzy system has been applied to all the three test functions with different dimensions. The experimental results illustrate that the fuzzy adaptive PSO is a promising optimization method, which is especially useful for optimization problems with a dynamic environment.

1 Introduction

Particle swarm optimization is a population-based evolutionary algorithm. It is similar to other population-based evolutionary algorithms in that the algorithm is initialized with a population of random solutions. It is unlike most of other population-based evolutionary algorithms (Goldberg 1989, Fogel 1994, Rechenberg 1994, Koza 1992), however, in that PSO is motivated by the simulation of social behavior instead of survival of the fittest, and each candidate solution is associated with a velocity (Eberhart and Kennedy 1995, Kennedy and Eberhart 1995, Eberhart, Dobbins and Simpson 1996, Kennedy 1997). The candidate solutions, called *particles*, then "fly" through the search space. The velocity is constantly adjusted according to the corresponding particle's experience and the particle's companions' experience. It is expected that the particles will move towards better solution areas. Mathematically, the particles are manipulated according to the following equation (Shi and Eberhart 1998a, 1998b):

$$v_{id} = w*v_{id} + c_1 * rand( ) * (p_{id} - x_{id}) + c_2 * Rand( ) * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

where c_1 and c_2 are positive constants, and rand() and Rand() are two random functions in the range [0,1]. $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ represents the ith particle. $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$ represents the best previous position (the position giving the best fitness value) of the ith particle. The symbol g represents the index of the best particle among all the particles in the population. $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ represents the rate of the position change (velocity) for particle i. Variable w is the inertia weight.

The balance between global and local search throughout the course of run is critical to the success of an evolutionary algorithm. In some implementations of evolutionary programming, balance between global and local search is obtained through adapting the variance (strategy parameter) of the Gaussian random function or step size. Furthermore, in some implementations, even the strategy parameter is encoded into the chromosomes to undergo evolution itself (Rochenberg 1994, Saravanan and Fogel 1996, Yao and Liu 1996). In PSO, it is the inertia weight that is used to balance the global and local search ability. The inertia weight has characteristics that are reminiscent of the temperature parameter in the simulated annealing (Eberhart and Shi 1998). A large inertia weight facilitates a global search while a small inertia weight facilitates a local search. By changing the

0-7803-6657-3/01/$10.00 ©2001 IEEE

inertia weight dynamically, the search ability is dynamically adjusted.

Since the search process of the PSO is non-linear and very complicated, it is hard, if not impossible, to mathematically model the search process to dynamically adjust the inertia weight. Instead, a fixed inertia weight (with or without additional noise) or a linearly decreasing inertia weight is deployed (Shi and Eberhart 1999, Eberhart and Shi 2001). By linearly decreasing the inertia weight from a relatively large value to a small value through the course of a PSO run, PSO tends to have more global search ability at the beginning of the run while having more local search ability near the end of the run. The simulation results on the benchmark problems illustrate that an inertia weight starting with a value 0.9 and linearly decreasing to 0.4 through the course of the run greatly improve the performance of PSO (Shi and Eberhart 1998a, 1999).

As mentioned above, the PSO search process is a nonlinear and complicated process and a linear decreasing inertia weight approach has a linear transition of search ability from global to local search, which does not truly reflect the actual search process required to find the optimum. This especially is true for dynamic optimization problems. For example, tracking problems (Eberhart and Shi 2001), where the environment itself is dynamically changed over the time, requires the search algorithm to have a nonlinear search ability to dynamically follow the changing environment. Therefore, for better performance, the inertia weight should be nonlinearly, dynamically changed to have better dynamics of balance between global and local search abilities. Due to the lack of knowledge of the search process, it is difficult, if not impossible, to design a mathematical model to adapt the inertia weight dynamically. Fortunately, over the years, some understanding of the PSO search process has been accumulated, and linguistic description of the search process is available. This understanding and linguistic description make a fuzzy system a good candidate for dynamically tuning the inertia weight of PSO. In (Shi and Eberhart 2000), a fuzzy system was designed to adjust the inertia weight of PSO for the Rosenbrock function. In this paper, the fuzzy system will be further modified to fit a wider range of optimization problems.

2 Fuzzy System

PSO, like other evolutionary algorithms, can be dynamically adapted on four different levels: environment, population, individual, and component level (Shi 2000). Since the inertia weight is a global variable in the equation (1) and it is applied to the whole population, the inertia weight adaptation discussed in this paper will be on the population level. To design a fuzzy system to dynamically adapt the inertia weight, normally, the inputs to the system are variables that measure the performance of the PSO, and the output of the system is the inertia weight or the change of the inertia weight (Shi 2000).

As in (Shi and Eberhart 2000), two variables are selected as inputs to the fuzzy system: the current best performance evaluation and the current inertia weight; the output variable is the change of the inertia weight.

The current best performance evaluation (CBPE) measures the performance of the best candidate solution found so far by the PSO. Different optimization problems have different ranges of performance measurement values. To design a fuzzy system with the CBPE as one of the inputs to be applicable to a wide range of optimization problems, the CBPE has to be converted into a normalized format. Assume, the optimization problems to be solved are minimization problems, and the estimated (or real) minimum is denoted as $CBPE_{min}$, and the non-optimal CBPE is denoted as $CBPE_{max}$. The non-optimal CBPE here represents that any solution with CBPE greater or equal to $CBPE_{max}$ is not an acceptable solution to the minimization problem. The normalized CBPE (NCBPE) can be calculated as

$$NCBPE = \frac{CBPE - CBPE_{\min}}{CBPE_{\max} - CBPE_{\min}} \qquad (3)$$

All three fuzzy variables (two input variables and one output variable) are defined to have three fuzzy sets: LOW, MEDIUM and HIGH with associated membership functions as *leftTriangle*, *Triangle* and *rightTriangle*, respectively. The definitions of these three membership functions are:

Left_triangle membership function:

$$f_{left\_triangle} = \begin{cases} 1 & if \quad x < x_1 \\ \dfrac{x_2 - x}{x_2 - x_1} & if \quad x_1 \le x \le x_2 \\ 0 & if \quad x > x_2 \end{cases}$$

Triangle membership function:

$$f_{triangle}(x) = \begin{cases} 0 & if\ x < x_1 \\ 2\dfrac{x - x_1}{x_2 - x_1} & if\ x_1 \le x \le \dfrac{x_2 + x_1}{2} \\ 2\dfrac{x_2 - x}{x_2 - x_1} & if\ \dfrac{x_2 + x_1}{2} < x \le x_2 \\ 0 & if\ x > x_2 \end{cases}$$

Right_triangle membership function:

$$f_{right\_triangle} = \begin{cases} 0 & if \quad x < x_1 \\ \dfrac{x - x_1}{x_2 - x_1} & if \quad x_1 \le x \le x_2 \\ 1 & if \quad x > x_2 \end{cases}$$

where x_1 and x_2 are critical parameters which determine the shape and location of the functions. Other membership function definitions are possible, of course, but the authors have found these to be useful for a variety of problems and to be easy to be implemented in microcontrollers and microprocessors. The whole fuzzy system for dynamically adapting the inertia weight can be described as in List 1 (Eberhart, Dobbins, and Simpson 1996, Shi, Eberhart and Chen 1999).

In List 1, the 9 on the first line means that there are 9 rules in the fuzzy system; the second line means that there are 2 inputs and 1 output in the fuzzy system. Following the second line is the definitions for the three variables. The first variable (input) is NCBPE, which has 3 fuzzy sets with a dynamic range (0,1). Each fuzzy set is associated with a membership function. The first one is a left triangle function with two critical parameters 0, 0.06, the second is a triangle function with two critical parameters 0.05, 0.4, and the final one is a right triangle with two critical parameters 0.3, 1. These four lines completely define the first input variable NCBPE (for a detailed description refer to (Eberhart, Dobbins, and Simpson 1996, Shi, Eberhart and Chen 1999)). The second input variable is the current inertia weight named *weight* and the output variable is the change (percentage) of the inertia weight, which is called *w_change*. Following the three variable definitions are the definitions of nine rules in the fuzzy system where the number 1 encodes fuzzy set LOW, 2 encodes MEDIUM and 3 encodes HIGH. For example, the first one 1 1 2 represents a fuzzy rule *if NCBPE is Low, and weight is Low, then the w_change is medium* (for details refer to (Eberhart, Dobbins, and Simpson 1996, Shi, Eberhart and Chen 1999)).

List 1: A description of a fuzzy system for adapting the inertia weight of PSO.

```
9
2 1
NCBPE 3 0 1
    leftTriangle 0 0.06
    Triangle 0.05 0.4
    rightTriangle 0.3 1

weight  3 0.2 1.1
    leftTriangle 0.2 0.6
    Triangle 0.4 0.9
    rightTriangle 0.6 1.1

w_change 3 -0.12 0.05
    leftTriangle -0.12 -0.02
    Triangle -0.04 0.04
    rightTriangle 0.0    0.05

1 1 2
1 2 1
1 3 1
2 1 3
2 2 2
2 3 1
3 1 3
3 2 2
3 3 1
```

3 Experimental Setting

For comparison, three benchmark functions reported in (Angeline 1998a, 1998b, Shi and Eberhart 1999), are used here. The first function is the Rosenbrock function described by

$$f(x)_1 = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \qquad (4)$$

The second function is the generalized Rastrigrin function described by equation (5):

$$f_2(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \qquad (5)$$

The last function is the generalized Griewank function described by equation (6):

$$f_3(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1 \qquad (6)$$

The three are all minimization problems. The $CBPE_{min}$ and $CBPE_{max}$ for the three functions are listed in Table 1.

Following the suggestion in (Fogel and Beyer 1995) and for the purpose of comparison, the asymmetric initialization method used in (Angeline 1998a) is adopted here for population initialization. Table 2 lists the initialization ranges of the three functions.

Table 1: $CBPE_{min}$ and $CBPE_{max}$ values

Function	$CBPE_{min}$	$CBPE_{max}$
f_1	0	500
f_2	0	70
f_3	0	0.15

Table 2: Asymmetric initialization ranges.

Function	Asymmetric Initialization Range
f_1	$(15,30)^n$
f_2	$(2.56,5.12)^n$
f_3	$(300,600)^n$

As in (Angeline 1998a) , for all the three functions, three different dimension sizes are tested: 10, 20 and 30. The maximum number of generations is set as 1000, 1500 and 2000 corresponding to the dimensions 10, 20 and 30, respectively. In order to investigate whether the PSO algorithm scales well or not, different population sizes are used for each function with different dimensions. They are population sizes of 20, 40, and 80. The general parameters of PSO are set as: $c_1=c_2=2$ for all the PSO runs. The V_{max} and X_{max} are listed in Table 3. A total of 50 runs for each experimental setting were conducted. The fuzzy system is designed to adapt the inertia weight by the authors according to the authors' understanding and experience with PSO which is given in List 1.

Table 3: X_{max} and V_{max} values

Function	X_{max}	V_{max}
f_1	100	100
f_2	10	10
f_3	600	600

4 Experimental Results and Discussion

Tables 4, 6 and 8 list the mean CBPE values of the best particle found for the 50 runs for the three benchmark functions with a linearly decreasing inertia weight (Shi and Eberhart 1999), respectively, and tables 5, 7, and 9 list the mean CBPE values of the best particle found for the 50 runs for the three benchmark functions using the fuzzy system described in List 1 for tuning the inertia weight.

Table 4: Mean CBPE values for the Rosenbrock function with linearly decreasing inertia weight.

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	96.1715
	20	1500	214.6764
	30	2000	316.4468
40	10	1000	70.2139
	20	1500	180.9671
	30	2000	299.7061
80	10	1000	36.2945
	20	1500	87.2802
	30	2000	205.5596

Table 5: Mean CBPE values for the Rosenbrock function with the fuzzy system tuning the inertia weight

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	66.01409
	20	1500	108.2865
	30	2000	183.8037
40	10	1000	48.76523
	20	1500	63.88408
	30	2000	175.0093
80	10	1000	15.81645
	20	1500	45.99998
	30	2000	124.4184

Table 6: Mean CBPE values for the Rastrigrin function with linearly decreasing inertia weight.

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	5.5572
	20	1500	22.8892
	30	2000	47.2941
40	10	1000	3.5623
	20	1500	16.3504
	30	2000	38.5250
80	10	1000	2.5379
	20	1500	13.4263
	30	2000	29.3063

Table 7: Mean CBPE values for the Rasgtrigrin function with the fuzzy system tuning the inertia weight

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	4.955165
	20	1500	23.27334
	30	2000	48.47555
40	10	1000	3.283368
	20	1500	15.04448
	30	2000	35.20146
80	10	1000	2.328207
	20	1500	10.86099
	30	2000	22.52393

Table 8: Mean CBPE values for the Griewank function with linearly decreasing inertia weight.

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	0.0919
	20	1500	0.0303
	30	2000	0.0182
40	10	1000	0.0862
	20	1500	0.0286
	30	2000	0.0127
80	10	1000	0.0760
	20	1500	0.0288
	30	2000	0.0128

Table 9: Mean CBPE values for the Griewank function with the fuzzy system tuning the inertia weight

Popu. Size	Dim.	Gene.	Mean Best Fitness
20	10	1000	0.091623
	20	1500	0.027275
	30	2000	0.02156
40	10	1000	0.075674
	20	1500	0.031232
	30	2000	0.012198
80	10	1000	0.068323
	20	1500	0.025956
	30	2000	0.014945

By comparing the results in Tables 4, 6, and 8 with the results in Tables 5, 7, and 9, respectively, it is easy to see that by using a fuzzy adaptive inertia weight the performance of PSO can be improved and have similar or better results than that of both PSO with a linearly decreasing inertia weight and the evolutionary programming method reported in (Angeline 1998a, Shi and Eberhart 1999). From the tables, it is also clear that PSO with different population sizes has reasonably similar performance. This feature is extremely important when the PSO is applied to real-time online problems where the time is critical and the solution evaluation consumes a significant amount of computation time. Furthermore, it can be seen from the tables that the performance of the PSO does not degrade a lot as the problem dimension scales up.

5 Conclusion

In this paper, a fuzzy system is implemented to dynamically adjust the inertia weight to improve the performance of the PSO. Three benchmark functions have been used for testing the performance of the fuzzy adaptive PSO. For comparison, simulations are conducted for both the fuzzy adaptive PSO and the PSO with a linearly decreasing inertia weight. The simulation results illustrate the performance of PSO is not sensitive to the population size, and the scalability of the PSO is acceptable. The results also illustrate that PSO with a fuzzy system tuning its inertia weight can improve its performance to some extent, at least for the three benchmark functions. More simulations are required to be run on other dynamic functions to illustrate the benefits and effectiveness of using a fuzzy adaptive PSO.

References

Angeline, P. J. (1998a). Evolutionary optimization versus particle swarm optimization: philosophy and performance difference. 1998 Annual Conference on Evolutionary Programming, San Diego.

Angeline, P. J. (1998b). Using selection to improve particle swarm optimization. IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

Eberhart, R. C., Dobbins, R. W., and Simpson, P. (1996), *Computational Intelligence PC Tools*, Boston: Academic Press.

Eberhart, R. C., and Kennedy, J. (1995). A new optimizer using particle swarm theory. Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.

Eberhart, R. C., Shi, Y. H. (1998). Comparison between genetic algorithms and particle swarm optimization. 1998 Annual Conference on Evolutionary Programming, San Diego.

Eberhart, R. C., Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center. (in press)

Fogel, D., Beyer H.-G. (1995). A note on the empirical evaluation of intermediate recombination. Evolutionary Computation, vol. 3, no. 4.

Fogel, L. J. (1994). Evolutionary Programming in Perspective: the Top-down View, in *Computational Intelligence: Imitating Life*, J.M. Zurada, R. J. Marks II, and C. J. Robinson, Eds., IEEE Press, Piscataway, NJ.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading MA: Addison-Welsey.

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. Proc. IEEE International Conference on Evolutionary Computation (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.

Kennedy, J., Eberhart, R. C. (1995). Particle swarm optimization. Proc. IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, pp. IV: 1942-1948.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

Rechenberg, I. (1994). Evolution Strategy, in *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks II, and C. Robinson, Eds., IEEE Press, Piscataway, NJ.

Saravanan, N., Fogel, D. (1996). An empirical comparison of methods for correlated mutations under self-adaptation. The Fifth Annual Conference on Evolutionary Programming.

Shi, Y. H. (2000). Fuzzy Adaptive Evolutionary Computation: A Review, The 4th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, July 23-26, 2000

Shi, Y. H., Eberhart, R. C. (1998a). Parameter selection in particle swarm optimization. 1998 Annual Conference on Evolutionary Programming, San Diego, March 1998.

Shi, Y. H., Eberhart, R. C., (1998b). A modified particle swarm optimizer. IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

Shi, Y. H., Eberhart, R. C. (1999). Empirical study of particle swarm optimization, 1999 Congress on Evolutionary Computation, Washington DC, USA, July 6-9.

Shi, Y. H., Eberhart, R. C. (2000) Experimental Study of Particle Swarm Optimization, The 4th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, July 23-26, 2000

Shi, Y. H., Eberhart, R. C., and Chen Y. (1999). Implementation of Evolutionary Fuzzy Systems, IEEE Transactions on Fuzzy Systems, Vol. 7, No. 2.

Yao, X., Liu, Y. (1996). Fast evolutionary programming. The Fifth Annual Conference on Evolutionary Programming.